**YouTube Clone – MERN Stack Capstone Project**

**Technologies Used**

**Frontend:**

* React
* TailwindCSS
* React Router DOM
* React Icons
* Vite

**Backend:**

* Node.js
* Express
* MongoDB (Mongoose)
* JWT authentication

**Tools & Hosting:**

* MongoDB Atlas
* ThunderClient/Postman (for testing APIs)
* Render/Netlify or similar (for deployment)

**1. What We Built**

This is a fully functional YouTube-like video web application where users can:

* Register and log in securely
* Create, customize, and manage their own channels
* Upload videos , with custom and auto thumbnails, video files only on local machine(for deployment soon implement URL method)
* Browse and search videos on the home page with category and search filters
* View detailed video pages, including comments (add/edit/delete, inline), like/dislike/share/download
* View channel pages with dynamic tabs (Home, Videos, Posts) and a matching subscriber count
* Manage personal uploads (only your uploads on your channel's video tab)
* Experience a UI and UX closely mimicking the real YouTube layout, with responsive design and protected routes

**2. How We Built It**

**Frontend Architecture**

* **Pages:** Home, Login, Register, VideoDetail, ChannelHome, ChannelVideos, NotFound
* **Components:** Header (app-wide, with blur effect), Sidebar, VideoCard, ChannelHeader, ProtectedRoute, CreateChannel, VideoUpload, NotificationAlert (centralized for all alerts/confirmations)
* **Routing:**
  + Dynamic routes for videos (/video/:id)
  + Channel pages (/channel/:username)
  + Channel videos tab (/channel/:username/videos)
  + Fallback error (/notfound)
* **Global State:**
  + AuthContext for login state
  + SearchContext and (optionally) ChannelContext for global features

**Backend API (Express + MongoDB)**

* **User Routes:** /register, /login, /users/:id, /channels/my..etc
* **Channel Routes:** /channels, /channels/my..etc
* **Video Routes:** /upload, /videos, /videos/:id, /videos/:id/comments..etc
* **Auth Middleware:** Protects routes using JWT with bcrypt for enhance security
* **File Uploads:** Videos and thumbnails handled through Multer with static public URLs, proper file extension handling
* **Mongoose Models:**
  + **User:** email, username, password, list of channel references
  + **Channel:** name, username, description, banner, reference to user, list of video references
  + **Video:** title, description, video file URL or YouTube URL, thumbnail URL, category, stats, references to user and channel, comments

**3. Why We Chose This Approach**

|  |  |
| --- | --- |
| Feature | Reason |
| React Context (AuthContext/ChannelContext) | Simple, flexible, and avoids prop drilling for global state management |
| ProtectedRoute Component | Keeps route protection logic dry and centralized |
| JWT Authentication with bcrypt | Simple, direct, and meets provided requirements |
| Cloud Database (MongoDB Atlas) | Quick setup, secure access, and scalable hosting |
| TailwindCSS | Fast prototyping, responsive by default, matches YouTube-like look easily |
| NotificationAlert Reusable Component | All confirmation dialogs and sign-in prompts use a uniform, center modal |
| Seed Script and Sample Data | Allows easy setup, and the "already imported" check prevents duplicate seeding |
| File Uploads with Multer | Securely stores real video and thumbnail files with correct URLs and MIME types |
| Realistic Comments CRUD | VideoDetail supports robust live add/edit/delete for comments, inline, no CSS change |

**4. Key Features Completed**

* Register/Login system using JWT and AuthContext
* Video uploading with real file streaming and custom/auto thumbnails
* Channel creation, management, and dynamic routing with subscriber count and proper isOwner logic
* Home page listing with filters and search (categories: All, Tech, Education, Music, etc.)
* Video detail page with actual video playback (file or YouTube), likes/dislikes, persistent comments (CRUD, real API)
* Unified Notification/Confirmation system for all actions (auth, delete, errors)
* Protected routes for upload, create channel, channel management
* Fully responsive UI and clean, minimal styling using TailwindCSS
* Error handling and fallback UI for invalid routes

**5. Learnings and Best Practices**

* Building a large UI from scratch teaches real-world responsive design and UI/UX flow
* Using React context and protected routes shows advanced full-stack architecture patterns
* Integrating a frontend SPA with a secure backend and cloud database demonstrates true full-stack capability
* JWT flows reinforce understanding of RESTful authentication patterns
* Real file handling, error modals, and centralized notifications are essential for real-world UX

**6. How to Run**

**Prerequisites**

* Node.js and npm
* MongoDB Atlas account

**Backend**

1. git clone [repo]
2. cd backend
3. npm install
4. Create .env with YOUR MongoDB URI, JWT\_SECRET, etc.
5. Run: npm run dev

**Frontend**

1. cd frontend
2. npm install
3. Set correct VITE\_API\_URL in .env
4. Run: npm run dev

App should run at localhost:5173 (frontend) and localhost:3000 (backend) by default.

**7. Conclusion**

This project demonstrates end-to-end full-stack skills: frontend and backend integration, authentication, media file handling, protected dynamic routing, and UI that closely matches an industry platform. The codebase is DRY, maintainable, and a strong base for adding more advanced social or video features in the future.